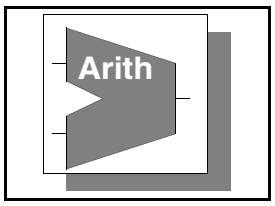
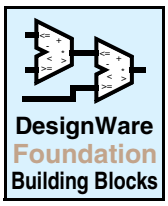




DW_div

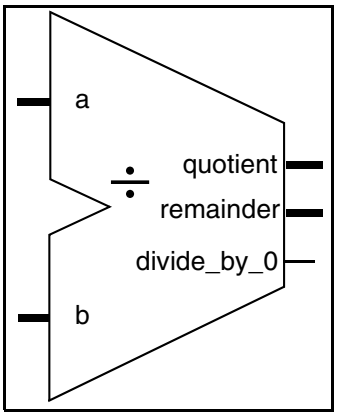
Combinational Divider

Version, STAR and Download Information: IP Directory

Features and Benefits

Revision History

- Parameterized word lengths
- Unsigned and signed (two's complement) data operation
- Remainder or modulus as second output
- Inferable using a function call
- Multiple architectures for area/performance trade-offs



Description

DW_div is a combinational integer divider with both `quotient` and `remainder` outputs. This component divides the dividend `a` by the divisor `b` to produce the `quotient` and `remainder`. Optionally, the `remainder` output computes the modulus.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
a	*a_width* bits	Input	Dividend
b	*b_width* bits	Input	Divisor
quotient	*a_width* bits	Output	Quotient
remainder	*b_width* bits	Output	Remainder / modulus
divide_by_0	1 bit	Output	Indicates if b equals 0

Table 1-2 Parameter Description

Parameter	Values	Description
a_width	≥ 1	Word length of `a`
b_width	≥ 1	Word length of `b`
tc_mode	0 or 1 Default: 0	Two's- complement control ■ 0: Unsigned ■ 1: Signed
rem_mode	0 or 1 Default: 1	Remainder output control ■ 0: a mod b ■ 1: a rem b, a % b

Table 1-3 Synthesis Implementations[a]

Implementation Name	Function	License Feature
rpl	Restoring ripple-carry divider synthesis model	None
cla	Restoring carry-look-ahead divider synthesis model	DesignWare
cla2	Radix-4 restoring carry-look-ahead divider synthesis model	DesignWare
cla3	Radix-8 restoring carry-look-ahead divider synthesis model	DesignWare
mlt[b]	Multiplicative divider synthesis model	DesignWare
lut[c]	Lookup-table-based divider synthesis model	DesignWare

a. During synthesis, Synopsys synthesis tools select the appropriate architecture for your constraints. Alternatively, you may use the set_implementation command to choose one specific implementation from those listed in this table. For details, see the documentation for your Synopsys synthesis tool.
b. The 'mlt' implementation is only useful when the divisor is small. So, this implementation is only valid when the *b_width* parameter (size of the divisor) is no greater than 10.
c. 1. The 'lut' implementation is only useful when the divider is small.
2. This implementation is only valid when (a_width + b_width) <= 12 and b_width <= 10.
3. The 'lut' implementation is supported beginning with Q-2019.12 in Fusion Compiler and Q-2019.12-SP1 in Design Compiler.

Table 1-4 Simulation Models

Model	Function
DW02.DW_DIV_CFG_SIM	Design unit name for VHDL simulation
dw/dw02/src/DW_div_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_div.v	Verilog simulation model source code

Functional Description

Table 1-5 Functional Description

tc_mode	rem_mode	a	b	quotient	remainder
0	0	a (unsigned)	b (unsigned)	int(a/b) (unsigned)	a mod b (unsigned)
0	1	a (unsigned)	b (unsigned)	int(a/b) (unsigned)	a rem b, a % b (unsigned)
1	0	a (two's complement)	b (two's complement)	int(a/b) (two's complement)	a mod b (two's complement)
1	1	a (two's complement)	b (two's complement)	int(a/b) (two's complement)	a rem b, a % b (two's complement)

The parameter *tc_mode* determines whether the data of the inputs `a` and `b` and the outputs `quotient` and `remainder` are interpreted as unsigned (*tc_mode* = 0) or two's-complement (*tc_mode* = 1) numbers.

The parameter *rem_mode* determines whether the output `remainder` corresponds to the division remainder (*rem_mode* = 1) or to the modulus (*rem_mode* = 0) as defined in the IEEE Standard 1076-1993 VHDL Language Reference Manual.

The modulus is different from the remainder only for two's complement division in VHDL. There, the modulus ("mod") has the sign of the divisor while the remainder ("rem") has the sign of the dividend. In Verilog, the modulus ("%") is actually the division remainder, which has the sign of the dividend.

The `quotient` and `remainder` outputs are defined as follows:

```
quotient = int (a/b)
```

For *rem_mode* = *1* (remainder):

```
remainder = a rem b = a – int (a/b) x b
```

If `a` and `b` have the same sign for *rem_mode* = *0* (modulus), then the following is true:

```
remainder = a mod b = a – int (a/b) x b
```

Otherwise, the following is true:

```
remainder = a mod b = a + ceil (|a/b|) x b
```

Example

7 rem 4 = 3 7 mod 4 = 3

7 rem –4 = 3 7 mod –4 = –1

–7 rem 4 = –3 –7 mod 4 = 1

–7 rem –4 = –3 –7 mod –4 = –3

Out-of-Range Results

In the case of dividing by zero, the `divide_by_0` output is set to 1 and the `quotient` is saturated to the maximum positive value if `a` is positive and to the minimum negative value if `a` is negative. The `remainder` is equal to `a`.

Example (two's complement):

```
"0010" / "0000" = "0111"
"0010" rem "0000" = "0010"

"1110" / "0000" = "1000"
"1110" rem "0000" = "1110"
```

In the case of dividing the minimum negative value by –1, the `quotient` would be one larger than the maximum representable positive value (overflow). The `quotient` is then set to the minimum negative value (corresponds to correct value in unsigned representation) and the `remainder` is set to 0. Note that this is the only case in which division of a negative number by a negative number does not result in a positive number.

Example (two's complement):

```
"1000" / "1111" = "1000”
"1000" rem "1111" = "0000"
```

Application Examples

Fixed-Point Arithmetic

For operands that are in integer.fraction format, the fraction width of the quotient is the fraction width of the dividend minus the fraction width of the divisor.

Example

```
110.01011 / 00100.101 = 000001.01
```

Rounding

DW_div truncates fractions to produce an integer quotient. To round to the nearest integer value, concatenate a fraction bit of zero to the least significant end of the dividend. The resulting quotient has a one-bit fraction. Add one-half to round the quotient to the nearest integer.

VHDL Example

```
signal quot_temp : unsigned(width downto 0);
quot_temp <= (a & '0') / b + 1;
quot <= quot_temp(width downto 1);
```

Verilog Example

```
wire frac;
assign {quot, frac} = DWF_div_uns ({a, 1'b0}, b) + 1;
```

Reciprocal

To compute the reciprocal of a number, simply divide 1.00...0 (represented by a one and `b_width-1` zeroes) by the number.

Division by a Normalized Number

When the divisor is normalized (its MSB is always one in unsigned numbers), DW_div will be better optimized if the MSB of the divisor is explicitly connected to a one (it is hard wired at the divider input) instead of being implicitly always one (it comes out of logic).

In two's complement normalized numbers, the second-most significant bit is always the inverse of the sign bit (MSB). There, best optimization results for DW_div are achieved by explicitly connecting the inverted sign bit to the second-most significant bit of the divisor.

VHDL Example

```
quot_uns <= a / ('1' & b(width-2 downto 0);
quot_tc <= a / (b(width-1) & not b(width-1) & b(width-3 downto 0));
```

Verilog Example

```
assign quot_uns = DWF_div_uns (a, {1'b1, b[width-2 : 0]);
assign quot_tc = DWF_div_tc a, {b[width-1], ~b[width-1], b[width-3 : 0]});
```

Unsigned-Signed Divider

DW_div can only perform either unsigned or signed division (specified by parameter *tc_mode*). If a divider is required that can do both (selectable by a `tc` pin), extend both inputs by one bit (zero-extend for unsigned numbers and sign-extend for two's complement numbers depending on the `tc` pin) and perform a signed division.

VHDL Example

```
signal quot_ext : signed(width downto 0);
quot_ext <= ((tc and a(width-1)) & a) / ((tc and b(width-1)) & b);
quot <= quot_ext(width-1 downto 0);
```

Verilog Example

```
wire ext_bit;
assign {ext_bit, quot} = DWF_div_tc ({tc & a[width-1], a},
                                     {tc & b[width-1], b});

    sc_uint< a_width >
        DW_div< a_width, b_width, tc_mode, rem_mode >::implement  (
            sc_uint< a_width >  a,
            sc_uint< b_width >  b,
            sc_uint< b_width > *  remainder,
            sc_uint< 1 > *  divide_by_0 )
```

Suppressing Warning Messages During Verilog Simulation

The Verilog simulation model includes macros that allow you to suppress warning messages during simulation.

To suppress all warning messages for all DWBB components, define the DW_SUPPRESS_WARN macro in either of the following ways:

- Specify the Verilog preprocessing macro in Verilog code:

  ```
  `define DW_SUPPRESS_WARN
  ```

- Or, include a command line option to the simulator, such as:

  `+define+DW_SUPPRESS_WARN` (which is used for the Synopsys VCS simulator)

The warning messages for this model include the following:

- If a divide-by-zero operation is attempted, the following message is displayed:

  ```
  WARNING: <instance_path>:
      at time = <timestamp>: Division by zero.
  ```

  To suppress this message, use the DW_SUPPRESS_WARN macro explained earlier.

Related Topics

- Math – Arithmetic Overview
- DesignWare Building Block IP User Guide

HDL Usage Through Operator Inferencing - VHDL

```
library IEEE, DWARE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use DWARE.DW_Foundation_arith.all;

entity DW_div_oper is
  generic ( width : positive := 8);
  port ( a             : in  std_logic_vector(width-1 downto 0);
         b             : in  std_logic_vector(width-1 downto 0);
         quotient_uns  : out std_logic_vector(width-1 downto 0);
         quotient_tc   : out std_logic_vector(width-1 downto 0);
         remainder_uns : out std_logic_vector(width-1 downto 0);
         remainder_tc  : out std_logic_vector(width-1 downto 0);
         modulus_uns   : out std_logic_vector(width-1 downto 0);
         modulus_tc    : out std_logic_vector(width-1 downto 0));
end DW_div_oper;

architecture oper of DW_div_oper is
begin

  -- operators for unsigned/signed quotient, remainder and modulus
  quotient_uns  <= unsigned(a) / unsigned(b);
  quotient_tc   <= signed(a) / signed(b);
  remainder_uns <= unsigned(a) rem unsigned(b);
  remainder_tc  <= signed(a) rem signed(b);
  modulus_uns   <= unsigned(a) mod unsigned(b);
  modulus_tc    <= signed(a) mod signed(b);

end oper;
```

HDL Usage Through Operator Inferencing - Verilog

```
module DW_div_oper (a, b, quotient_uns, quotient_tc, remainder_uns,
                    remainder_tc);
  parameter width = 8;

  input [width-1 : 0] a, b;
  output [width-1 : 0] quotient_uns, remainder_uns;
  output signed [width-1 : 0] quotient_tc, remainder_tc;
  // operators for unsigned/signed quotient and remainder
  assign quotient_uns = a / b;
  assign quotient_tc = $signed(a) / $signed(b);
  assign remainder_uns = a % b;
  assign remainder_tc = $signed(a) % $signed(b);
endmodule
```

HDL Usage Through Function Inferencing - VHDL

```
library IEEE, DWARE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use DWARE.DW_Foundation_arith.all;

entity DW_div_func is
  generic ( width : positive := 8);
  port ( a             : in  std_logic_vector(width-1 downto 0);
         b             : in  std_logic_vector(width-1 downto 0);
         quotient_uns  : out std_logic_vector(width-1 downto 0);
         quotient_tc   : out std_logic_vector(width-1 downto 0);
         remainder_uns : out std_logic_vector(width-1 downto 0);
         remainder_tc  : out std_logic_vector(width-1 downto 0);
         modulus_uns   : out std_logic_vector(width-1 downto 0);
         modulus_tc    : out std_logic_vector(width-1 downto 0));
end DW_div_func;

architecture func of DW_div_func is
begin
  -- function calls for unsigned/signed quotient, remainder and modulus
  quotient_uns  <= std_logic_vector(DWF_div (unsigned(a), unsigned(b)));
  quotient_tc   <= std_logic_vector(DWF_div (signed(a), signed(b)));
  remainder_uns <= std_logic_vector(DWF_rem (unsigned(a), unsigned(b)));
  remainder_tc  <= std_logic_vector(DWF_rem (signed(a), signed(b)));
  modulus_uns   <= std_logic_vector(DWF_mod (unsigned(a), unsigned(b)));
  modulus_tc    <= std_logic_vector(DWF_mod (signed(a), signed(b)));
end func;
```

HDL Usage Through Function Inferencing - Verilog

```
module DW_div_func (a, b, quotient_uns, quotient_tc, remainder_uns,
                    remainder_tc, modulus_uns, modulus_tc);

  parameter width = 8;

  input  [width-1 : 0] a;
  input  [width-1 : 0] b;
  output [width-1 : 0] quotient_uns;
  output [width-1 : 0] quotient_tc;
  output [width-1 : 0] remainder_uns;
  output [width-1 : 0] remainder_tc;
  output [width-1 : 0] modulus_uns;
  output [width-1 : 0] modulus_tc;

  // pass "a_width" and "b_width" parameters to the inference functions
  parameter a_width = width;
  parameter b_width = width;

  // Please add search_path = search_path + {synopsys_root + "/dw/sim_ver"}
  // to your .synopsys_dc.setup file (for synthesis) and add
  // +incdir+$SYNOPSYS/dw/sim_ver+ to your verilog simulator command line
  // (for simulation).
  `include "DW_div_function.inc"

  // function calls for unsigned/signed quotient, remainder and modulus
  assign quotient_uns  = DWF_div_uns (a, b);
  assign quotient_tc   = DWF_div_tc (a, b);
  assign remainder_uns = DWF_rem_uns (a, b); // corresponds to "%" in Verilog
  assign remainder_tc  = DWF_rem_tc (a, b);  // corresponds to "%" in Verilog
  assign modulus_uns   = DWF_mod_uns (a, b); // corresponds to "mod" in VHDL
  assign modulus_tc    = DWF_mod_tc (a, b);  // corresponds to "mod" in VHDL

endmodule
```

HDL Usage Through Component Instantiation - VHDL

```
library IEEE, DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DW_Foundation_comp_arith.all;

entity DW_div_inst is

  generic ( width    : positive := 8;
            tc_mode  : natural  := 0;
            rem_mode : natural  := 1);
  port ( a           : in  std_logic_vector(width-1 downto 0);
         b           : in  std_logic_vector(width-1 downto 0);
         quotient    : out std_logic_vector(width-1 downto 0);
         remainder   : out std_logic_vector(width-1 downto 0);
         divide_by_0 : out std_logic);
end DW_div_inst;

architecture inst of DW_div_inst is
begin
  -- instance of DW_div
  U1 : DW_div
    generic map ( a_width => width, b_width => width,
                  tc_mode => tc_mode, rem_mode => rem_mode)
    port map ( a => a, b => b,
               quotient => quotient, remainder => remainder,
               divide_by_0 => divide_by_0);
end inst;

-- pragma translate_off
configuration DW_div_inst_cfg_inst of DW_div_inst is
  for inst
  end for;
end DW_div_inst_cfg_inst;
-- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_div_inst (a, b, quotient, remainder, divide_by_0);

  parameter width    = 8;
  parameter tc_mode  = 0;
  parameter rem_mode = 1; // corresponds to "%" in Verilog

  localparam a_width = width;  // With numerator (a) the same
  localparam b_width = width;  // size as the divisor (b) both
                               // a_width and b_width parameters
                               // for DW_div are the same

  input  [a_width-1 : 0] a;
  input  [b_width-1 : 0] b;
  output [a_width-1 : 0] quotient;
  output [b_width-1 : 0] remainder;
  output                 divide_by_0;

  // Please add +incdir+$SYNOPSYS/dw/sim_ver+ to your verilog simulator
  // command line (for simulation).

  // instance of DW_div
  DW_div #(.a_width(a_width), .b_width(b_width),
           .tc_mode(tc_mode), .rem_mode(rem_mode))
    U1 (.a(a), .b(b),
        .quotient(quotient), .remainder(remainder),
        .divide_by_0(divide_by_0));
endmodule
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
June 2023	DWBB_202212.4	■ Adjusted parameter names in the example in "HDL Usage Through Component Instantiation - Verilog" on page 11
December 2022	DWBB_202212.0	■ Added the 'lut' implementation in Table 1-3 on page 2
July 2020	DWBB_201912.5	■ Added "Suppressing Warning Messages During Verilog Simulation" on page 6 ■ Added this Revision History table and the document links on this page

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
www.synopsys.com